**DOCUMENTATION MANUAL FOR** **DEVELOPMENT OF A TIMETABLE SHUFFLER APPLICATION FOR DMI**

**UNIVERSITY**

**Table of Contents**

1. **Introduction**

* 1.1 Overview
* 1.2 Technologies Used

**2. Setting Up the Development Environment**

* 2.1 Prerequisites
* 2.2 Installation Steps

**3. Creating the Electron Application**

* 3.1 Initializing the Project
* 3.2 Setting Up Electron
* 3.3 Creating the Main Process

**4. Setting Up Express for Backend**

* 4.1 Initializing Express
* 4.2 Creating API Endpoints

**5. Implementing the Database**

* 5.1 Structuring the Database
* 5.2 CRUD Operations

**6. Creating the User Interface**

* 6.1 HTML Structure
* 6.2 Styling with CSS
* 6.3 JavaScript Interactions

**7. Integrating Frontend and Backend**

* 7.1 Making API Calls
* 7.2 Handling Responses

**8. Testing the Application**

* 8.1 Manual Testing
* 8.2 Automated Testing

**9. Deployment**

* 9.1 Packaging the Application
* 9.2 Distributing the Application

**10. Conclusion**

**11. References**

**1. Introduction**

**1.1 Overview**

This manual provides step-by-step instructions to build a user management system using Electron for the desktop interface and Express for the backend API. The application will support functionalities like user registration, login, and timetable management for students and lecturers.

**1.2 Technologies Used**

* Electron: A framework for building cross-platform desktop applications using web technologies (HTML, CSS, and JavaScript).
* Express: A web application framework for Node.js that provides a robust set of features for web and mobile applications.
* Node.js: A JavaScript runtime built on Chrome's V8 engine, allowing server-side JavaScript execution.
* JSON: A lightweight data interchange format used for storing user data persistently.

**2. Setting Up the Development Environment**

**2.1 Prerequisites**

Ensure you have the following installed:

* Node.js: Download and install from [Node.js official site](https://nodejs.org/).
* npm (Node Package Manager): Included with Node.js.
* A code editor: Such as Visual Studio Code or any preferred editor.
* Git: For version control.

**2.2 Installation Steps**

1. Open a terminal and create a project directory:

*```bash*

*mkdir electron-project*

*cd electron-project*

**2. Initialize npm:**

*```bash*

*npm init -y*

*```*

**3. Creating the Electron Application**

**3.1 Initializing the Project**

1. Install Electron:

*```bash*

*npm install electron --save-dev*

*```*

**3.2 Setting Up Electron**

1. **Create `main.js**` for the main process:

*```javascript*

*const { app, BrowserWindow } = require('electron');*

*function createWindow() {*

*const win = new BrowserWindow({*

*width: 800,*

*height: 600,*

*webPreferences: {*

*nodeIntegration: true,*

*contextIsolation: false*

*}*

*});*

*win.loadFile('index.html');*

*}*

*app.whenReady().then(createWindow);*

*```*

**3.3 Creating the Main Process**

**1. Modify `package.json` to include a start script:**

*```json*

*"scripts": {*

*"start": "electron ."*

*}*

*```*

**2. Create a basic HTML file named `index.html`.**

**4. Setting Up Express for Backend**

**4.1 Initializing Express**

1. Install Express and body-parser:

```bash

npm install express body-parser

```

2. Create `server.js` for the server setup.

**4.2 Creating API Endpoints**

1. Define API routes in `server.js` to handle user registration and login.

**5. Implementing the Database**

**5.1 Structuring the Database**

1. Create `database.json` to store user information:

*```json*

*{*

*"students": {},*

*"lecturers": {},*

*"rooms": {},*

*"modules": {},*

*"batches": {}*

*}*

*```*

**5.2 CRUD Operations**

**1. Implement CRUD operations in your API routes as necessary**.

**6. Creating the User Interface**

**6.1 HTML Structure**

1. Extend `index.html` to include forms for user registration and login.

**6.2 Styling with CSS**

1. Create a CSS file `styles.css` for styling your application.

**6.3 JavaScript Interactions**

1. Create a JavaScript file `renderer.js` for handling user interactions.

**7. Integrating Frontend and Backend**

**7.1 Making API Calls**

1. Use the `fetch` API to communicate with the Express backend for user registration and login.

**7.2 Handling Responses**

1. Implement response handling to inform users about success or failure.

**8. Testing the Application**

**8.1 Manual Testing**

1. **Run the server** and **start the Electron app** to test functionalities.

**8.2 Automated Testing**

1. Use testing frameworks like Jest or Mocha for unit testing.

**9. Deployment**

**9.1 Packaging the Application**

1. Install Electron Packager to create executables:

*```bash*

*npm install electron-packager --save-dev*

*```*

**9.2 Distributing the Application**

1. **Add a packaging script** in your `package.json` and run it to build your application.

**10. Conclusion**

Congratulations! You have built an Electron application integrated with an Express backend. You can further enhance your application by adding more features and refining the user interface.

11. References

- [Electron Documentation](https://www.electronjs.org/docs/latest)

- [Express Documentation](https://expressjs.com/)

- [Node.js Documentation](https://nodejs.org/en/docs/)